Proactive Wrangling: Mixed-Initiative End-User Programming of Data Transformation Scripts

Philip J. Guo[*], *Sean Kandel*[*], *Joseph M. Hellerstein*[†], *Jeffrey Heer*[*]

[*]Stanford University
{pg, skandel, jheer}@cs.stanford.edu

[†]University of California, Berkeley
hellerstein@cs.berkeley.edu

ABSTRACT

Analysts regularly wrangle data into a form suitable for computational tools through a tedious process that delays more substantive analysis. While interactive tools can assist data transformation, analysts must still conceptualize the desired output state, formulate a transformation strategy, and specify complex transforms. We present a model to proactively suggest data transforms which map input data to a relational format expected by analysis tools. To guide search through the space of transforms, we propose a metric that scores tables according to type homogeneity, sparsity and the presence of delimiters. When compared to "ideal" hand-crafted transformations, our model suggests over half of the needed steps; in these cases the top-ranked suggestion is preferred 77% of the time. User study results indicate that suggestions produced by our model can assist analysts' transformation tasks, but that users do not always value proactive assistance, instead preferring to maintain the initiative. We discuss some implications of these results for mixed-initiative interfaces.

ACM Classification: H5.2 [Information interfaces and presentation]: User Interfaces – Graphical user interfaces.

General terms: Design, Human Factors

Keywords: Data transformation, data cleaning, end-user programming, mixed-initiative interfaces, data analysis

INTRODUCTION

The increasing scale and accessibility of data — including government records, web data, and system logs — provides an under-exploited resource for improving governance, public policy, organizational strategy, and even our personal lives [23]. However, much of this data is not *suitable* for use by analysis tools. Data is often stored in idiosyncratic formats or designed for human viewing rather than computational processing (e.g., cross-tabulations within spreadsheets).

These hurdles require that analysts engage in a tedious process of data transformation (or *data wrangling*) to map input data to a form consumable by downstream tools. Both prior work [3] and our own conversations with analysts indicate that wrangling is one of the most time-consuming aspects of analysis. As a result, domain experts regularly spend more time manipulating data than they do exercising their specialty, while less technical audiences may be excluded.

Analysts typically wrangle their data by writing scripts in programming languages such as Python or Perl or by manually editing the data within spreadsheets. To assist this process, researchers have developed novel interactive tools. Potter's Wheel [21] and Google Refine [11] are menu-driven interfaces that provide access to common data transforms. Wrangler [13] extends this approach by (a) automatically suggesting applicable transforms in response to direct manipulation of a data table and (b) providing visual previews to aid transform assessment. Each of these tools enable skilled practitioners to rapidly specify transformation workflows; however, they fall short in helping users formulate data transformation strategies. Given an input data set, what is the desired output state? What operations are possible and which sequence of operations will result in suitable data?

One complication is that there is no single "correct" output format. A data layout amenable to plotting in Excel is often different than the format expected by visualization tools such as Tableau. That said, one format is required by many database, statistics and visualization tools: a relational data table. In this format, each row contains a single data record and each column represents a variable of a given data type.

In this paper, we augment the Wrangler transformation tool [13] to aid transform discovery and strategy formation. We take a mixed-initiative [10] approach to end-user programming: we generate **proactive suggestions** to improve the suitability of a data set for downstream tools. In other words, we want the computer to help analysts get data into the format that computers expect. We make three contributions:

A model of data table *"suitability"* that enables generation of proactive transform suggestions. We introduce a metric that indicates the degree to which a table adheres to a relational format usable by downstream tools. We use this metric to guide automated search through the space of possible operations. As the search space is too large to evaluate at interactive rates, we couple our metric with empirically-derived heuristics for pruning the space of candidate transforms.

An analysis of algorithm behavior assessing the strengths and limitations of our suggestions. Across a set of realistic transformation tasks, we find that our method suggests over

UIST'11, October 16-19, 2011, Santa Barbara, CA, USA.

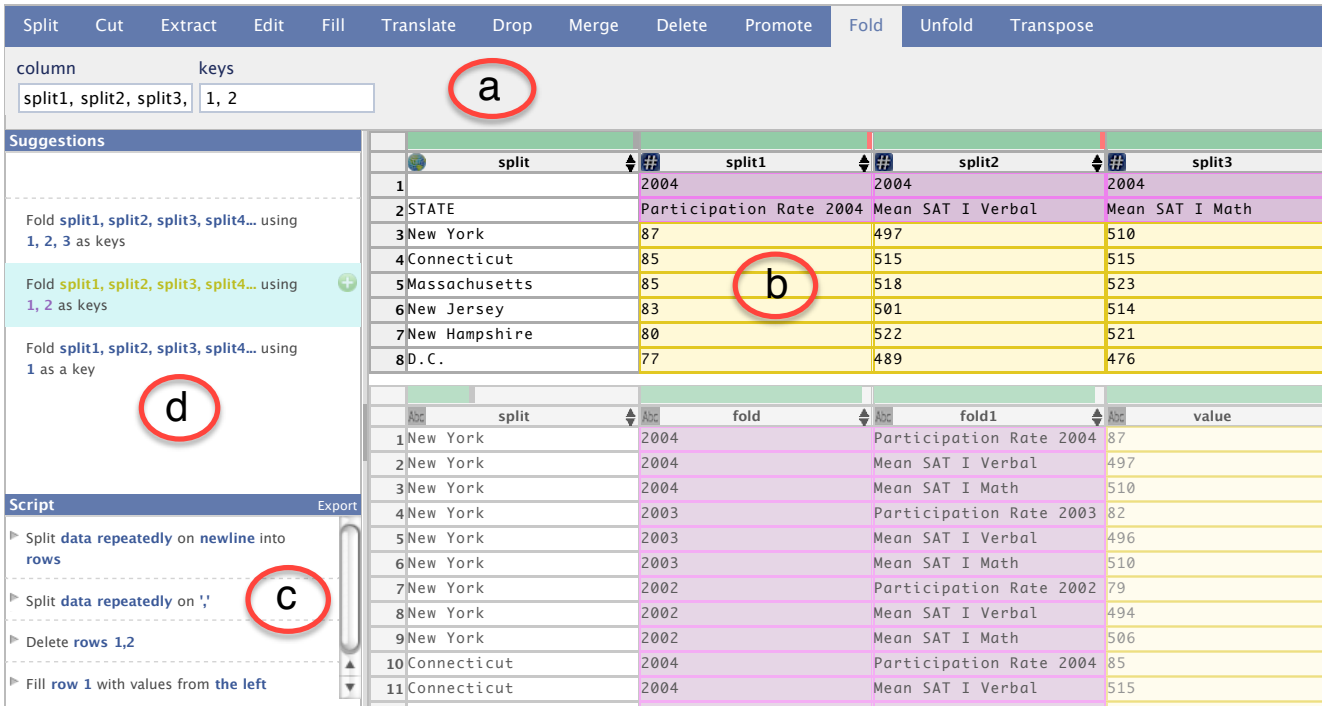

Figure 1: The Wrangler User Interface. Clockwise from the top: (a) tool bar for transform specification, (b) data table display, (c) history viewer containing an exportable transformation script, (d) suggested transforms. The effect of the selected *Fold* transform is previewed in the data table display using before (top) and after (bottom) views.

half (53%) of the steps used in "ideal" transformations. In these cases our suggestions exhibit high precision: the top-ranked suggestion is the preferred option 77% of the time.

A user study examining how end users apply proactive suggestions. We find that our method can help users complete transformation tasks, but not always in the manner that we expected. Many users express a desire to maintain control and roundly ignore suggestions presented prior to interaction with the data table. However, users do not express similar concerns when suggestions generated by our proactive model are inserted among "reactive" suggestions specific to an initiating interaction. User assessment of suggestions appears to improve when users see themselves as the initiators.

We also present a redesign of the Wrangler interface to better surface the space of available data transforms and parameters. The resulting interface enables manual specification or refinement of transforms and facilitates learning from the examples provided by automated suggestions.

RELATED WORK

Researchers have proposed numerous strategies for simplifying programming tasks, including domain-specific languages [18], visual programming [1], keyword programming [17], and programming-by-demonstration [2]. Myers et al. [20] provide a survey of these approaches. Here, we review prior work in end-user programming tools for data transformation.

Menu-Driven Data Transformation

Both database and HCI researchers have devised graphical interfaces [6, 11, 21] for reformatting and cleaning data. Each of these tools share a similar design: a data table view and a set of menu-accessible transformation operators. Google Refine [11] combines menu-driven commands with faceted histograms for data profiling and filtering. Ajax [6] similarly provides an interface for transform specification with advanced facilities for entity resolution. Potter's Wheel [21] is a graphical interface for authoring statements in a declarative transformation language and offers extensible facilities for data type inference. Each of these tools enable skilled practitioners to transform data more rapidly. However, each imposes a learning curve: users must learn both the available operations and how to combine them to achieve common data transformation needs.

Programming-by-Demonstration Systems

Another class of targeted data manipulation tools rely on programming by demonstration (PBD) techniques [2]. Potluck [12] uses simultaneous editing [19] to help users perform data integration. Karma [22] infers text extractors and transformations for web data from examples entered in a table. Vegemite [16] extends CoScripter [15] to integrate web data, automate the use of web services, and generate shareable scripts. Dontcheva et al. [4] use PBD techniques to enable extraction, integration, and templated search of web data. Gulwani [8] introduces an algorithm for learning expressive string processing programs from input-output examples. PADS [5] infers nuanced data parsers from a set of positive examples. Though well-suited for their intended tasks, these systems are insufficient for the more general demands of data wrangling. For example, while text extraction and mass editing are quite valuable for data transformation, the above tools lack other needed operations such as table reshaping, aggregation, and missing value imputation.

Closely related to our current work is Harris and Gulwani's [9] system for learning table transformations from an example input-output pair. Given input and output tables, the system can infer a program that filters, duplicates, and/or reorganizes table cells to generate the output table. These resulting programs can express a variety of table reshaping operations.

Our work differs in multiple respects. First, the approach of Harris and Gulwani treats table cells as atomic units, and thus has limited expressiveness. For example, it does not support transformations involving text editing or extraction. Second, their approach requires that a user specify both input and output examples, and thus know the details of the desired output in advance. In the case of large tables, creating the output example may be tedious. On the other hand, providing a small example from which the proper transformation can be inferred may require significant insight. Third, the actual mechanics of the resulting transformation are opaque, potentially limiting user skill acquisition. In contrast, our approach does not accept an output example but instead leverages the assumption that the user desires a relational data table with homogeneous columns. This enables our system to offer proactive suggestions in the context of a full-featured data transformation environment.

DATA WRANGLER

In this work, we introduce extensions to *Data Wrangler* [13], a system that combines multiple end-user programming strategies to facilitate specification of data transformation scripts. Underlying Wrangler is a *declarative domain-specific language* that includes a comprehensive set of transformation operators and enables code-generation of reusable programs (e.g., Python and JavaScript code). Wrangler provides natural language descriptions and *visual previews* with which users can assess and verify transform behaviors. Wrangler uses *programming-by-demonstration methods* (c.f., [8, 19]) to specify regular expression patterns and row predicates. In addition, Wrangler generates *automatic transform suggestions* in response to user interactions with a data table.

The Wrangler Transformation Language

The transformation scripts produced by Wrangler consist of a linear sequence of individual transforms, expressed in an underlying declarative language. The transforms support common data cleaning and reformatting tasks such as splitting or extracting text values, deleting or merging columns, batch text editing, interpolating values, and reorganizing cell layout (table *reshaping*). Table 1 summarizes the most common transforms surfaced in the user interface. The language is the direct descendant of the Potter's Wheel language [21] and draws on concepts introduced in SchemaSQL [14]. This prior work [14, 21] also provides formal proofs of the language's expressiveness, showing it is capable of expressing any one-to-one or one-to-many transform of cell values.

In addition, the declarative nature of the language facilitates implementation across a variety of platforms. Thus the Wrangler tool can generate executable code for multiple runtimes, including Python and JavaScript. For example, we often wrangle a data subset within the user interface and then export a resulting Python script to transform much larger databases (millions or more rows) on a server machine.

Transform	Description
Cut	Remove selected text from cells in specified columns.
Delete	Remove rows that match given indices or predicates.
Drop	Remove specified columns from the table.
Edit	Edit the text in each cell of the specified columns.
Extract	Copy text from cells in a column into a new column.
Fill	Fill empty cells using values from adjacent cells.
Fold	Reshape a table into columns of key-value sets; selected rows map to keys, selected columns to values.
Merge	Concatenate multiple columns into a single column.
Promote	Promote row values to be the column names.
Split	Split a column into multiple columns by delimiters.
Translate	Shift the position of cell values by a given offset.
Transpose	Transpose the rows and columns of the table.
Unfold	Reshape a table by mapping key-value sets to a collection of new columns, one per unique key.

Table 1: The Wrangler Transformation Language. Each transform accepts as parameters some combination of enumerable values and text, row, or column selection criteria. For further discussion, see [13, 21].

	Boys	Girls
Australia	1	2
Austria	3	4
Belgium	5	6
China	7	8

Australia	Boys	1
Australia	Girls	2
Austria	Boys	3
Austria	Girls	4
Belgium	Boys	5
Belgium	Girls	6
China	Boys	7
China	Girls	8

Figure 2: An example of table reshaping. A *Fold* operation transforms the table on the left to the one on the right; an *Unfold* operation performs the reverse.

A critical feature of the language design is its compactness: with only a dozen operators, analysts can complete a wide variety of data cleaning tasks. Moreover, most operators accept identical parameter types, namely row, text and column selections. The compact language design is intended to facilitate exploration and learning by limiting the number of possible operations considered. An additional benefit is that this reduced set of formally-defined operators lends itself to computational search over operation sequences.

The Wrangler User Interface

The Wrangler user interface (shown in Figure 1) allows analysts to specify transforms in the underlying language. The interface contains four primary components: along the top of the display is a tool bar for transform specification, the right panel contains an interactive data table display, and the left panel contains both automated transform suggestions (top) and an interactive history viewer describing the current script (bottom). Similar to other transformation tools [11, 21], a transform can be specified manually by selecting a transform type from the tool bar and then entering in desired parameter values. However, Wrangler provides additional facilities to aid transform specification and verification.

Automated Transform Suggestions

Wrangler's data table display supports a set of common interactions, such as selecting rows and columns by clicking their

headers or selecting and manipulating text within table cells. While these actions can be used to specify parameters for a selected transform type, manual specification can be tedious and requires that users have a firm command of the available transforms. To facilitate rapid specification, Wrangler infers applicable transforms directly from an initiating interaction.

After the user makes a selection on the data table, Wrangler generates automatic suggestions in a three-phase process. First, Wrangler infers a set of possible parameter values in response to the user's selection. Example parameters include regular expressions matching selected text substrings and row predicates matching selected values, generated using programming-by demonstration methods [13, 19]. Second, Wrangler enumerates a list of transforms that accept the inferred parameters. Third, Wrangler filters and ranks the resulting transforms according to historical usage statistics and heuristics intended to improve result quality. These transforms then appear as a ranked list of suggestions in the interface, where users can preview their effects and modify their parameters. For more details regarding the Wrangler inference engine, we refer interested readers to Kandel et al. [13].

Assessing Transform Effects

When creating data transformation scripts, users often find it difficult to understand a transform's effect prior to executing it [13]. To aid in transform assessment, Wrangler presents suggestions using natural language descriptions to aid quick scanning of the suggestions list. When a transform is selected, Wrangler uses visual previews to enable users to quickly evaluate its effects without executing it. When feasible, Wrangler displays in-place previews in the data table, drawing users' attention to the effect of the transform in its original context. For complex reshaping operations, Wrangler shows before and after images of the table and uses color coding to help users perceive the correspondence of cells between the two states (Figure 1b).

Design Issues and Opportunities

Wrangler's automated suggestions and visual assessment features help analysts iteratively hone in on a desired transformation. A user study by Kandel et al. [13] finds that these mechanisms can accelerate specification and facilitate discovery of applicable transforms. However, the initial Wrangler design also suffers from some shortcomings.

Surfacing Transform Parameters

While natural language descriptions enable users to scan suggestions, they provide little guidance for how to specify parameters manually. This disconnect constitutes a missed opportunity: in addition to accelerating specification, suggested transforms might serve as *instructive examples* of more advanced system use. The initial Wrangler design also buries transforms within hierarchical menus, limiting visibility.

In response, we redesigned the Wrangler user interface to better surface available transforms and their parameters. The tool bar along the top of the user interface makes all transforms immediately visible, and the parameters for a current transform appear directly below. As users preview suggested transforms, the complete parametrization is now visible in the tool bar. We apply linked highlighting between parameters in natural language descriptions and those in the tool bar. In terms of Green's Cognitive Dimensions of Notation [7], this redesign is intended to improve the *visibility* of options and decrease the *viscosity* of manually tuning inferred parameters. In terms of Horvitz's principles for mixed-initiative interaction [10], the design provides a *shared working memory* by which users can more directly observe (and thus hopefully learn from) system-initiated actions.

Specifying Transforms and Formulating Strategies

Observing user activity with Wrangler reveals that analysts may have difficultly formulating data cleaning strategies [13]. Novice users are often unsure which target data state is the one best suited for subsequent analysis. Even when the target data state is known, users are often unsure which sequence of operations is needed to reach that state. We have observed that both novice and expert users may consequently resort to blind exploration of the available transforms. This process is hampered by the fact that some operations remain difficult to specify despite the aid of automated suggestions. In particular, reshaping operations that significantly alter the arrangement of table cells are hard to conceptualize and apply. However, it is exactly these operations that are often necessary to map input data to a format usable by analysis tools.

We attempt to address these issues by extending Wrangler to provide *proactive suggestions* in addition to "reactive" suggestions initiated by user actions. Our proactive suggestions are intended to lead users towards effective cleaning strategies and facilitate the specification of complex reshaping operations. After describing the details of our proactive suggestion algorithm, we go on to analyze the behavior of our algorithm and evaluate our approach in a user study.

PROACTIVE DATA WRANGLING

We now describe extensions to Wrangler to automatically generate proactive suggestions. In addition to the previous method of suggesting transforms in response to user-initiated actions (e.g., selecting a row or column), *Proactive Wrangler* continually analyzes the current state of the data table and provides suggestions to make the table more suitable for import into a relational database or analytic tool. These suggestions are intended to both accelerate the work of experts and guide novices towards a desired outcome. For example, we wish to surface complex reshaping operations (*fold* and *unfold*) within a "recognition" task, rather than a harder "recall" task in which the user needs to determine which operation they require and manually initiate its specification.

A Proactive Data Cleaning Scenario

To give a sense of how proactive suggestions ease the data cleaning process, Figure 3 illustrates a complete scenario where a user transforms data imported from an Excel spreadsheet (Table 1 within Figure 3) into a dense relational format (Table 6). First, Wrangler analyzes the initial table state and makes 3 proactive suggestions. The user ignores them and instead manually selects and deletes the first two rows, turning Table 1 into Table 2. Then the user chooses to execute the 2nd suggestion (shown in **bold**), *Split column 2 on ':'*, which transforms the data into Table 3. Wrangler continues proactively suggesting additional transforms, and the user can simply pick the desired ones from the list.

1.

Bureau of I.A.	
Regional Director	Numbers
Niles C.	Tel: (800)645-8397
	Fax: (907)586-7252
Jean H.	Tel: (918)781-4600
	Fax: (918)781-4604
Frank K.	Tel: (615)564-6500
	Fax: (615)564-6701

Manually select and delete the first two rows

Proactive suggestions (none chosen):
1. Fill column 1 from above
2. Split column 2 on ':'
3. Delete empty rows

2.

Niles C.	Tel: (800)645-8397
	Fax: (907)586-7252
Jean H.	Tel: (918)781-4600
	Fax: (918)781-4604
Frank K.	Tel: (615)564-6500
	Fax: (615)564-6701

Proactive suggestions:
1. Fill column 1 from above
2. Split column 2 on ':'
3. Delete empty rows

3.

Niles C.	Tel	(800)645-8397
	Fax	(907)586-7252
Jean H.	Tel	(918)781-4600
	Fax	(918)781-4604
Frank K.	Tel	(615)564-6500
	Fax	(615)564-6701

Proactive suggestions:
1. Delete empty rows
2. Fill column 1 from above

4.

Niles C.	Tel	(800)645-8397
	Fax	(907)586-7252
Jean H.	Tel	(918)781-4600
	Fax	(918)781-4604
Frank K.	Tel	(615)564-6500
	Fax	(615)564-6701

Proactive suggestions:
1. Fill column 1 from above

5.

Niles C.	Tel	(800)645-8397
Niles C.	Fax	(907)586-7252
Jean H.	Tel	(918)781-4600
Jean H.	Fax	(918)781-4604
Frank K.	Tel	(615)564-6500
Frank K.	Fax	(615)564-6701

Proactive suggestions:
1. Unfold column 1 on column 3
2. Unfold column 2 on column 3

6.

	Tel	Fax
Niles C.	(800)645-8397	(907)586-7252
Jean H.	(918)781-4600	(918)781-4604
Frank K.	(615)564-6500	(615)564-6701

Result: cleaned table ready for DB import

Figure 3: Using Proactive Wrangler to reformat raw data from a spreadsheet. The proactive suggestion chosen at each step is highlighted in **bold**. In the Wrangler UI, proactive suggestions are displayed in the suggestions panel (Figure 1d).

Although the above example is small, it is indicative of a common usage pattern when cleaning data with Proactive Wrangler. In particular, cleaning often involves executing a mix of human-initiated and proactively-suggested transforms. Here the user must recognize that rows 1 and 2 are extraneous headers that should be deleted, but Wrangler proactively suggests all other necessary transforms. In the remainder of this section, we describe how Wrangler generates, filters and ranks these proactive suggestions.

A Metric for Data Table "Suitability"

We define a *suitable* data table as one that can be loaded and effectively manipulated by an analytic tool or relational database. These tools often expect data in a relational format where each row contains a single data record and each column represents a variable of a given data type. Although the raw data in Table 1 of Figure 3 is human-readable, it is not suitable for analytic tool or database import because of its irregular structure: it contains extraneous header rows, empty rows, and telephone/fax numbers are located in cells mixed with other strings, separated by a colon delimiter (e.g., "Tel: (800)645-8397"). In contrast, the compact relational result in Table 6 is more usable by downstream tools.

We created a metric that assesses how amenable a table is for import into downstream tools. Our metric rewards column type homogeneity (H), fewer empty values (E), and lack of delimiters (D). A *lower* score indicates a more suitable table. A homogeneous, dense, delimiter-free table (e.g., Table 6 in Figure 3) has a "perfect" score of zero; tools expecting relational tables should be able to import it.

For a table T with rows R and columns C, we define the table suitability S as

$$S(T) = \left(1 - \frac{\sum_{c \in C} H_c(T)}{|C|}\right) + \frac{E(T) + D(T)}{|R|\,|C|} \qquad (1)$$

where $|R|$ is the number of rows, $|C|$ is the number of columns, and each component function is defined as follows.

H_c is the **homogeneity** of column c, the sum of squares of the proportions of each data type $Type$ present in that column:

$$H_c = \sum_{Type} \left(\frac{|i \in R : c_i \in Type|}{|R|}\right)^2 \qquad (2)$$

Wrangler parses each cell's contents into the most specific possible type, which can be a built-in like *number* or *date*, a user-defined type like *zip code* or *country name*, or a generic *string* type for those that fail to match a more specific type (for more details regarding Wrangler's type inference, see Kandel et al. [13]). If 75% of values in a column are numbers and 25% are dates, then the column's homogeneity is $(0.75)^2 + (0.25)^2 = 0.625$. If 50% are numbers and 50% are dates, then the column is less homogeneous: $(0.5)^2 + (0.5)^2 = 0.5$. A column that contains only values of a single type has the maximum possible homogeneity score of 1. Our table suitability score averages the homogeneity of all columns and subtracts that average from 1, so that higher homogeneity leads to a *lower* (more "suitable") score.

E is the count of **empty cells**. We favor denser tables, as missing values may indicate a sub-optimal data layout possibly containing replicated values in adjacent columns.

D is the count of **delimiters**. A delimiter is a comma, colon, pipe, or tab character. We favor tables with fewer delimiters, as those tables are more likely to consist of atomic data rather than compound elements that could split into specific values.

Our suitability score is invariant of table size, since its terms are all normalized by the numbers of rows and columns. However, one of its limitations is that it does not penalize *data loss*. In the extreme pathological case, deleting all cells except for one can lead to a perfect score of 0 (assuming that last cell contains no delimiters). In the next section, we describe how we designed Proactive Wrangler to avoid suggesting such undesirable transforms.

Poor scores in each component of Equation 1 make a table ill-suited for tool import. In databases and analytic tools, a column with a mixture of data types typically can only be imported as uninterpreted strings. A column with many delimiters may be treated as strings rather than composite values of more specific types. A column with many empty cells often indicates replicated values in adjacent columns. All these situations can limit downstream analyses.

Generating Candidate Transforms

Proactive Wrangler automatically suggests transforms that improve a table's suitability score by first generating candidate suggestions, and then evaluating, filtering, and ranking these candidates before presenting them to the user.

One way to avoid suggesting undesirable lossy transforms is to explicitly penalize data loss when ranking suggestions; however, this suffers from efficiency problems because all possible suggestions must still be generated, evaluated, and ranked. Instead, we adopt a simpler approach: we prune the space of suggestions to ensure that Proactive Wrangler never makes suggestions that involve significant data loss. We limit lossy transforms to deletions of mostly-empty rows (which often appear in spreadsheet headers or footnotes). We contend that data deletion should be left to the user's discretion, and so Proactive Wrangler should only help to compact (fill and delete empty cells), parse (split on delimiters), and reshape (fold and unfold) the table. Thus, it generates the following types of transforms as candidates at each step:

- **Fold** columns $C_1, \ldots, C_n$ using rows $R_1, \ldots, R_n$ as keys
- **Unfold** column C_1 on column C_2
- **Split** column C into multiple columns using a delimiter (comma, colon, pipe, or tab)
- **Fill** all empty cells of row R with values from the left
- **Fill** all empty cells of column C with values from above
- **Delete** all mostly-empty rows ($\geq 75\%$ empty cells)
- **Delete** all empty rows
- **Delete** all empty columns

Since most of these transforms take row and/or column indices as parameters, enumerating all possibilities is both infeasible and undesirable. Assuming that a table has $|R|$ rows and $|C|$ columns, there are $O(|R|)$ possible row fills, $O(|C|)$ possible column fills and splits, $O(2^{|R|} \cdot 2^{|C|})$ possible folds, and $O(|C|^2)$ possible unfolds. It would take far too much time to evaluate all possible parameterizations, precluding interactive response rates. Even if efficiency were not a concern, many parameterizations are undesirable since they trigger degenerate cases of certain transforms.

Informed by our data cleaning experiences and empirical observations from a corpus of public data sets (described in the next section), we developed rules to reduce the set of transform parameters and eliminate undesirable candidates:

Fill. Proactive Wrangler only generates *fill* candidates if the majority ($\geq 50\%$) of cells in a given row or column are empty. This type of transform is useful for filling in sparse key rows (or columns) before doing a fold (or unfold).

Fold. Proactive Wrangler only generates *fold* candidates using all columns except for the leftmost one (i.e., Columns 2 through $|C|$), and using at most the first 3 rows as keys (i.e., Row 1, Rows 1–2, or Rows 1–3).

The above rule reduces the number of possible folds from $O(2^{|R|} \cdot 2^{|C|})$ to 3 and covers the majority of observed use cases for transforming a cross-tabulation into a relational format. To quantify our intuition, we manually inspected all 114 tables in the 2010 OECD Factbook, an annual publication of world socioeconomic data sets from `oecd.org`. For all except for 2 of the tables, the *only* sensible fold used columns 2 through $|C|$, and for all except for 1 table, the only sensible fold used at most the first 3 rows as keys. A cursory inspection of cross-tabulation data sets from Wikipedia, US Bureau of Justice, and `data.gov` showed no tables that could be sensibly folded using columns other than 2 through $|C|$ or using more than the first 3 rows as keys.

Also, Proactive Wrangler only generates fold candidates if the leftmost column contains all unique and non-null values in rows other than the key row(s); otherwise the folded table will have conflicts. For example, if there were a duplicate "Australia" row in the table on the left of Figure 2, then the results of a fold would be ambiguous.

Unfold. Proactive Wrangler only generates *unfold* candidates of the form "Unfold C_1 on C_2" if all columns except for C_2 are completely homogeneous (single-typed) and non-empty; otherwise the headers in the resulting cross-tab will not be well-typed. (The values in C_2 become the data matrix at the core of the resulting table, so it can have empty cells.) Also, Proactive Wrangler only generates unfold candidates if the table has exactly 3, 4, or 5 columns, which corresponds to 1, 2, or 3 key columns, respectively. Our intuition for imposing this limit is similar to why we only restrict folds to using at most the first 3 rows as keys: it covers the majority of use cases, as indicated by our data corpus.

The aforementioned rules reduce the number of candidate transforms from $O(2^{|R|} \cdot 2^{|C|})$ to $O(|R| + |C|)$. At each step (e.g., in Figure 3), Proactive Wrangler generates all of these candidates at once and then evaluates, filters, and ranks them before presenting them to the user.

Ranking and Presenting Proactive Suggestions

After Proactive Wrangler generates a set of candidate transforms, it executes each of them on the current table and calculates the suitability score for the resulting table. It removes all candidates that result in tables with scores higher ("worse") than the current table. We use a slightly modified scoring method for *unfold* transforms. The operation "Unfold C_1 on C_2" transforms the values in column C_2 into a 2D data matrix at the core of the resulting table. Thus, that matrix should be scored as an atomic unit rather than as disparate columns. Proactive Wrangler calculates the score improvement of an unfold by the change in homogeneity and density between C_2 and the resulting data matrix.

Proactive Wrangler then presents the remaining candidates to the user in the suggestions panel (Figures 1d and 3), sorted by the amount of improvement ($-\Delta S$). A user can preview and execute one of these proactive suggestions or can choose to ignore them, and instead make a selection on the data table to surface the ordinary context-dependent suggestions.

CHARACTERIZING PROACTIVE ALGORITHM BEHAVIOR

To assess the quality of our proactive suggestions, we analyzed the behavior of our algorithm across a collection of diverse data sets. For each, we compared the suggestions produced by our algorithm with "ideal" transformation scripts constructed by hand. As there is often more than one feasible "clean" goal state, we strove to transform each data table into a relational format amenable for import into a database or analytic tool and also sought to remove extraneous tokens from fields (e.g., parentheses, leading dashes, etc.).

We compared 20 distinct data sets in our analysis. While this collection is relatively small, each data set was chosen because it is representative of a much larger class of data we have collected. For example, although we only included three tables from the 2010 OECD Factbook of world socioeconomic data, all 114 tables from that source had a format that matched one of these three canonical tables. Although we cannot claim that our corpus is a representative sample of all data that users might want to clean, we strove to select data of diverse origins, formats, and complexity:

Origins. Our corpus consists of a mix of government and NGO data (e.g., OECD, US Department of Justice, data.gov), scientific data (e.g., flower classifications, audiology experiments) and data scraped from websites (e.g., Wikipedia reference tables, Craigslist apartment rental listings).

Format. Data formats range over cross-tabulations (e.g., left half of Figure 2), Excel reports intended for human consumption (e.g., Table 1 in Figure 3), line-oriented records with custom token separators and ad-hoc formats generated by scientific experiments.

Complexity. The most concise cleaning script we constructed for data sets in our corpus ranged from 1 to 7 transforms in length (i.e., some data sets were noticeably "dirtier" than others). The mean script length was 3.6 transforms.

We now summarize the results of using Proactive Wrangler to transform the 20 data sets in our corpus. When available, we always selected the highest-ranking proactive suggestion that would lead us towards the goal state; otherwise we initiated a selection on the data table and chose to execute the appropriate context-specific suggestion. Of all executed transforms, 53% (39/73) were proactive suggestions; the remaining 47% (34/73) resulted from human-initiated actions. When appropriate proactive suggestions appeared, the top-ranked (#1) suggestion was the preferable choice 77% of the time (30/39). The lowest rank of any chosen proactive suggestion was 6, and the mean rank was 1.6.

For reference, in the ideal case 100% of all executed transforms would come from the top-ranked proactive suggestion. In reality, our scoring metric is not perfect and there are certain classes of transforms that Proactive Wrangler is designed specifically *not* to offer. For our corpus, the following transforms required human intervention: splitting on non-standard delimiters, extracting and cutting substrings, and deleting non-sparse rows and columns. Across our collection, 6 of the corpus tables can be cleaned solely using proactive suggestions, 9 require a mix of human-initiated selections and proactive suggestions, and 5 require exclusively human selections. We now discuss each category in turn.

Fully Proactive

Six of the data tables in our corpus (30%) exhibited the best-case performance for Proactive Wrangler: all of the required transforms appear as proactive suggestions. This typically occurs for tables whose individual elements are properly formatted but the table itself requires re-structuring.

For example, all 114 tables in the 2010 OECD Factbook and all 50 state-level crime data sets from the Department of Justice are cross-tabulations embedded within Excel spreadsheets. Even though we only included a few samples in our corpus, the rest of the tables from those sources are identically formatted. For all of these tables, Proactive Wrangler suggests to delete the mostly-empty title and footnote rows, fill in the header key row(s), and then perform the appropriate fold to get the data into a relational format.

Hybrid

In the common case (9/20 tables, 45%), data must be cleaned using a combination of user-initiated transforms and proactive suggestions. For these kinds of tables in our corpus, 55% of executed transforms (23/42) came from proactive suggestions. By design, Proactive Wrangler does not offer certain types of suggestions, namely of transforms that lead to data loss. This limitation causes Wrangler to sometimes get "stuck" at a local optimum point when it cannot generate any more proactive suggestions. In those cases, human intervention can allow Wrangler to escape local optima and resume generating useful proactive suggestions.

For example, in a veteran's hospital dataset, we first chose a proactive suggestion to delete all empty rows, but then we had to manually delete an unnecessary column and remove (cut) the trailing '%' character from all numbers representing percentages. After those user-initiated clean-ups, Wrangler proactively offers the proper fold suggestion, which we executed to complete our cleaning task.

Manual

Wrangler cannot offer useful proactive suggestions for data that require extensive string parsing within individual cells, which occurred in a quarter of our examples (5/20 tables, 25%). These data sets all require the user to perform custom string splits, cuts, or extracts. For two of these data sets, Wrangler "stayed out of the way" and did not offer any proactive suggestions. For the other three, Wrangler offered on average two (unhelpful) proactive suggestions. Our findings from the user study (see next section) indicate that users find it easy to ignore such unhelpful suggestions.

For example, each line in a Craigslist apartment listings dataset is a string in this format:

```
$2475/2br - Superb location - (palo alto) pic
```

To create a table of rental prices, bedroom count, and locations, one can split each line on the '/' character, cut the '$' from the price to make it into an integer, and extract the location (e.g., "palo alto") from within the parentheses. These types of string manipulations are easier for a human to initiate using Wrangler's reactive suggestions; it is difficult for an algorithm to suggest these manipulations *a priori* with both high precision and recall. Consequently, Wrangler does not currently attempt to make proactive suggestions for them.

USER STUDY

We conducted a user study comparing versions of Wrangler with and without proactive suggestions. We originally hypothesized that proactive suggestions would simplify specification of commonly-used, but hard-to-specify reshaping transforms (e.g., fold, unfold), while causing limited distraction to users specifying other types of transforms. Furthermore, we hypothesized that by lowering the cost of finding applicable transforms, users would more quickly complete data cleaning tasks requiring multi-step transformations.

As we will discuss, our results indicate that suggestions generated by our proactive model can help users discover applicable transforms, but not always in the manner we anticipated. Users often dismissed proactive suggestions presented prior to an initiating interaction, but made use of those exact same suggestions when presented later in the session.

Participants

We recruited 16 participants (11 male, 5 female) who had never used Wrangler before. All were university students with at least moderate experience programming and using data analysis tools such as Microsoft Excel or Matlab; all came from Computer Science or related majors. Participants received a $15 gift certificate for one hour of their time.

Methods

We evaluated the impact of proactive suggestions by comparing Proactive Wrangler to a baseline version that only generates suggestions in reaction to user-initiated interactions. First, we guided each subject through a 15-minute tutorial of Wrangler, highlighting both proactive and reactive modes.

Next, we had each subject perform data cleaning tasks on four data sets that were miniature versions of data sets from our corpus. We designed each task so that, in proactive mode, helpful proactive suggestions were available at most but not all steps (the "Hybrid" category from the algorithm behavior study). We imposed a 10-minute limit per task. The four cleaning tasks were divided into two pairs, each requiring structurally similar transformations. We gave participants a paper printout of the desired goal table state for each task. For a given pair of tasks, we assigned subjects one task using Proactive Wrangler and the other task using the baseline "reactive" Wrangler. We counterbalanced task order and Wrangler versions across subjects.

After completing the four tasks, we asked subjects to describe their preferences for reactive versus proactive suggestions. Each study session was administered on a 15" MacBook Pro laptop and recorded using screen capture software.

The first eight subjects used a version of Proactive Wrangler that displayed proactive suggestions only when there was no active selection on the data table; as soon as the user makes a selection, the proactive suggestions were replaced by the standard reactive suggestions. However, we observed that most subjects largely ignored the proactive suggestions in this configuration. After interviewing those subjects about their rationale, we updated Wrangler to append the top three proactive suggestions to the list of reactive suggestions normally generated from a user interaction. In other words, Wrangler now displayed proactive suggestions regardless of how the user interacted with it. This modified *embedded proactive* interface was seen by the last eight subjects.

Results

We initially expected subjects to complete tasks faster using Proactive Wrangler, but in fact we found no significant difference in completion times between proactive, embedded, or reactive variants ($F_{2,51} = 0.059$, $p = 0.943$ according to repeated measures ANOVA). We compared within individual tasks and controlled for the effects of task order, but found no significant differences. The average task completion time was 231 seconds; in only one instance did a user fail to complete a task by going over the 10-minute time limit.

Upon reviewing the usage logs, we found that subjects made sparing use of proactive suggestions. Most users (11/16, 69%) executed a proactive suggestion at least once. However, the total number of such transforms was small: in proactive conditions, subjects executed 1.16 proactive suggestions per task on average, compared to an average of 4.9 reactive suggestions on those same tasks.

To understand why subjects were reluctant to use the proactive suggestions, we reviewed our recorded experiment sessions to identify critical incidents. We identified three recurring patterns: subjects ignoring useful proactive suggestions, previewing but dismissing suggestions, and viewing suggestions as a last resort. We now discuss each.

Ignoring proactive suggestions

Nearly one-third of the subjects (5/16) simply never previewed any proactive suggestions, despite having used them in the tutorial. All of these subjects ignored at least one proactive suggestion that they later executed after making a selection and choosing an *identical* reactive suggestion. For example, after 137 seconds of work, one subject was almost

done with a task and only needed to perform a final unfold. The necessary unfold was displayed as the sole proactive suggestion, but the subject ignored it and spent the next 86 seconds experimenting with selecting columns and specifying various folds and unfolds. He finally chose the correct unfold and completed his task, at which point he said, *"A proactive suggestion would have really helped me here!"*

Once we integrated proactive suggestions among reactive suggestions, 4 out of the 8 subjects who used the embedded version ignored a useful proactive suggestion only to click on it immediately after selecting an unrelated part of the table. One subject wanted to do a fold when in fact he needed an unfold (which was proactively suggested but ignored). When he selected some columns and attempted a fold, he noticed the proactive unfold suggestion at the bottom of the list and realized he needed to unfold, so he selected it and completed his task. This unfold would not have surfaced if he had not been using the embedded proactive interface.

Previewing, then ignoring, proactive suggestions

Almost half of our subjects (7/16) actually previewed the effects of a useful proactive suggestion, dismissed it, and a short time later made a selection and executed the *identical* reactive suggestion. As a typical example, one subject previewed a proactively-suggested unfold transform for a few seconds. But instead of executing it, he immediately selected the two columns in the table necessary to have Wrangler reactively suggest the *same* unfold. He then navigated directly to that unfold transform, previewed, and executed it.

Proactive suggestions as a last resort

Three subjects used proactive suggestions only when they ran out of ideas for triggering reactive suggestions. For example, out of the 167 seconds that one subject spent on a task, she was stuck staring at the table for 60 seconds (36% of total time) before she looked at the top-ranked proactive fold, previewed, and then executed it. This was the first (and only) time that she paused for long enough to actually consider using proactive suggestions; for the remainder of her session, she was continually making selections on the table, which surfaced reactive suggestions and hid the proactive ones.

Study Limitations

All subjects had some programming background, and this might make them more likely to want to explore rather than immediately follow proactive suggestions. One subject remarked, *"Maybe it's just me, but I like to try [initiating Wrangler actions] myself to see how they work."*

We provided specific target end states to subjects and timed them. Although many analysts have some end state in mind when working with data, discovering an appropriate end state is often an exploratory task. Several subjects reported that given a more ambiguous task, they believe they would be more likely to explore proactive suggestions, since these may inform what types of end states are possible.

Summary

Our user study largely disconfirms our original hypotheses. Users initially ignored proactive suggestions and on average did not complete tasks more quickly with them. However, proactive suggestions did aid the transformation process. Over two-thirds of subjects executed a proactive suggestion at least once. Suggestions generated by our proactive model proved helpful when embedded among reactive suggestions and when users failed to complete tasks by other means. At minimum, proactive suggestions are occasionally helpful and do not disrupt users. Moreover, our results raise some future research questions for the design of mixed-initiative interfaces. We discuss these in the next section.

DISCUSSION

In this paper, we presented a metric for table "suitability" that we apply to generate suggested data transforms. When compared with idealized transformation scripts, our model produces helpful suggestions for a majority of transform steps and ranks the preferred suggestion highly. In our own use, we have found proactive suggestions to be of great assistance: we regularly use the tool for data wrangling and find proactive suggestions to be a valuable complement. As a result, we believe our proactive suggestion model helps advance the state-of-the-art in interactive data transformation.

However, our user study results cast a shadow on our initial motivations. We find little evidence that proactive suggestions help novice users complete transformation tasks more quickly or learn data cleaning strategies. Upon review of our notes and recordings, we hypothesize multiple factors shape the observed usage patterns: *attentional blindness* to proactive suggestions, users' desire to *sustain initiative*, and *insufficient expertise* to recognize the value of suggestions.

Attention Blindness

Some subjects reported that they did not attend to proactive suggestions, even after using them in the tutorial. Many instead clicked on the table to initiate reactive suggestions. On the one hand, subjects expressed appreciation that proactive suggestions did not interrupt their workflow: annoying distractions are a classic complaint for mixed-initiative interfaces. However, insufficient visibility also seems to be a problem. Wrangler and similar tools might benefit from cues that emphasize suggested operations. For instance, subtle animation or colored backgrounds might draw attention to high-confidence suggestions without interrupting users' flow.

User Agency and Sustained Initiative

Many users expressed a pre-existing disdain for proactive interfaces. One said: *"I hate suggestions popping up at me on the computer ... I just want to get them out of my way."* Users described a distrust of automated suggestions in productivity software, citing examples like the infamous Microsoft Office 'paper clip'. Another user, describing his reluctance to use Wrangler's proactive suggestions, stated *"I knew what I wanted to do with the data, so I just did it myself."* Ironically, users voiced no such qualms regarding reactive suggestions, which were regularly employed to transform data.

During the first half of our study, only 4/8 subjects (50%) executed proactive suggestions. After we deployed the embedded proactive interface, 7/8 subjects (88%) executed suggestions generated by our proactive model, including those inserted into reactive suggestions. It is possible that user assessment of suggested transforms improves when users per-

ceive of themselves as the initiator. We observed several instances of subjects making a selection, choosing an embedded proactive suggestion, and executing it *after* the same suggestion had already been proactively presented and viewed.

Subjects seem to value suggestions more when they are offered in response to an initiating action, even if the suggestions are generated independently. While speculative, this observation suggests that future research might examine not only the utility of suggestions to the stated task goal [10], but also how interface design and turn-taking affects user receptiveness to those suggestions. What design decisions might help sustain users' sense of control in mixed-initiative UIs?

Expertise for Transform Assessment

Our own internal use of Proactive Wrangler and the results of our study with novice users are at a disconnect. In our experience, proactive suggestions can reduce the cognitive effort and also the time required to specify transformations. Initially we hypothesized that proactive suggestions can speed transformation by casting transform specification as a "recognition" task, in which users recognize a desired transform among the list of suggestions. Our user study results suggest otherwise. Prior knowledge of transformation strategies may be needed to rapidly "recognize" the value of a transform. Thus, proactive suggestions may actually be more effective for expert users than for novices. New preview mechanisms might better facilitate user assessment of suggestions. For example, thumbnails of a transform's effects might be easier to comprehend than the current textual descriptions.

More sophisticated approaches may be needed to foster learning of multi-step transformation strategies. In most cases subjects followed a greedy approach to discovering an appropriate chain of transformations. Some complex transformations may require steps in which it is difficult to gauge progress from an intermediate state (e.g., a fold transform followed by a subsequent unfold). Subjects stated that they were worried that proactive suggestions might lead them down the wrong path. However, they often did later select and execute transforms that were identical to those previously suggested by Proactive Wrangler. By this point the subject had explored more alternatives and used interactions to communicate their intent. It is unclear which of these two factors gave the user more trust in the system.

Ultimately, users may be the most successful once they are able to articulate and evaluate their own transformation strategies. Towards this aim, future work might examine alternative approaches to fostering expertise. Tutorial generation may be one means. Another possibility is to examine the effects of multi-step suggestions. New user interface techniques for suggesting and previewing multiple steps may aid these situations; can a different design prompt multi-step reasoning rather than greedy search? More broadly, designing mixed-initiative interfaces to foster expertise through example presents a compelling challenge for future work.

ACKNOWLEDGMENTS

The authors thank all the participants in the user study. This research was supported in part by NSF grant CCF-0964173, the Boeing Company, and Greenplum/EMC.

REFERENCES

1. M. Burnett. Visual programming. In J. G. Webster, editor, *Encyclopedia of Electrical and Electronics Engineering*. John Wiley & Sons Inc., 1999.
2. A. Cypher. *Watch What I Do: Programming by Demonstration*. MIT Press, 1993.
3. T. Dasu and T. Johnson. *Exploratory Data Mining and Data Cleaning*. John Wiley & Sons, Inc., New York, NY, 2003.
4. M. Dontcheva, S. M. Drucker, D. Salesin, and M. F. Cohen. Relations, cards, and search templates: user-guided web data integration and layout. In *ACM UIST*, pages 61–70, 2007.
5. K. Fisher and D. Walker. The Pads project: An overview. In *International Conference on Database Theory*, March 2011.
6. H. Galhardas, D. Florescu, D. Shasha, and E. Simon. Ajax: an extensible data cleaning tool. In *ACM SIGMOD*, page 590, 2000.
7. T. R. G. Green. Cognitive dimensions of notations. In *British Computer Society, Human-Computer Interaction Specialist Group on People and Computers V*, pages 443–460. Cambridge University Press, 1989.
8. S. Gulwani. Automating string processing in spreadsheets using input-output examples. In *ACM POPL*, pages 317–330, 2011.
9. W. Harris and S. Gulwani. Spreadsheet table transformations from examples. In *ACM PLDI*, 2011.
10. E. Horvitz. Principles of mixed-initiative user interfaces. In *ACM CHI*, pages 159–166, 1999.
11. D. Huynh and S. Mazzocchi. Google Refine. `http://code.google.com/p/google-refine/`.
12. D. F. Huynh, R. C. Miller, and D. R. Karger. Potluck: semi-ontology alignment for casual users. In *ISWC*, pages 903–910, 2007.
13. S. Kandel, A. Paepcke, J. Hellerstein, and J. Heer. Wrangler: Interactive visual specification of data transformation scripts. In *ACM CHI*, 2011.
14. L. V. S. Lakshmanan, F. Sadri, and S. N. Subramanian. SchemaSQL: An extension to SQL for multidatabase interoperability. *ACM TODS*, 26(4):476–519, 2001.
15. G. Leshed, E. M. Haber, T. Matthews, and T. Lau. CoScripter: automating & sharing how-to knowledge in the enterprise. In *ACM CHI*, pages 1719–1728, 2008.
16. J. Lin, J. Wong, J. Nichols, A. Cypher, and T. A. Lau. End-user programming of mashups with Vegemite. In *IUI*, pages 97–106, 2009.
17. G. Little and R. C. Miller. Translating keyword commands into executable code. In *ACM UIST*, pages 135–144, 2006.
18. M. Mernik, J. Heering, and A. M. Sloane. When and how to develop domain-specific languages. *ACM Comput. Surv.*, 37(4):316–344, 2005.
19. R. C. Miller and B. A. Myers. Interactive simultaneous editing of multiple text regions. In *USENIX Tech. Conf.*, pages 161–174, 2001.
20. B. A. Myers, A. J. Ko, and M. M. Burnett. Invited research overview: end-user programming. In *ACM CHI Extended Abstracts*, pages 75–80, 2006.
21. V. Raman and J. M. Hellerstein. Potter's wheel: An interactive data cleaning system. In *Proc. VLDB*, 2001.
22. R. Tuchinda, P. Szekely, and C. A. Knoblock. Building mashups by example. In *ACM IUI*, pages 139–148, 2008.
23. H. Varian. Hal Varian on how the Web challenges managers. In *McKinsey Quarterly*, Jan 2009.